CS 6410: Compilers

Fall 2019

HW 4 – Static semantics, attribute grammar and type checking

Assigned: Tuesday, October 14, 2019, Due: Saturday, November 2, 2019

Instructor: Tamara Bonaci
College of Computer and Information Science
Northeastern University – Seattle

- Please submit your homework as a single .pdf file through Canvas.
- You do not have to type in your submission - hand-written and then scanned, or photographed documents are fine, as long as the total size of your document is not too big, and your document is readable.
- This assignment is meant to be worked on individually, and you should submit it by **11:59pm on Saturday, November 2, 2019**.

Problem 1

(a) Compute the FIRST, FOLLOW and Nullable for the following grammar:

```
0. exp' ::= exp $
1. exp ::= id
2. exp ::= ( exp )
3. exp ::= ( type ) exp
4. type ::= id
```

(b) Compute the FIRST, FOLLOW and Nullable for the following grammar:

```
0. eqns' ::= eqns $ ($ is the end-of-file marker)
1. eqns ::= eq sup eqns
2. eqns ::= eq
3. eq::=x
```

Problem 2 (Cooper and Torczon, Problem 5.12)

You are writing a compiler for a simple lexically-scoped language. Consider the example shown in the listing below.

(a) Draw the symbol table and its content at line 11.
(b) What actions are required for symbol table management when the parser enters a new procedure, and when it exists it?

```
1.  procedure main
2.    integer a, b, c:
3.    procedure f1(w, x)
4.      integer a, x, y;
5.      call f2(w, x);
6.    end;
7.    procedure f2(y,z)
8.      integer a, y, z;
9.      procedure f3(m, n)
10.       integer b, m, n;
11.       c = a * b * m * n;
12.     end;
13.     call f3(c, z);
14.   end;
15.   ...
16    call f1(a, b);
17. end;
```

Problem 3 (Cooper and Torczon, Problem 4.7)

A Pascal program can declare two integers variables `a` and `b` with the syntax:
`var a, b: int`
This declaration might be described with the following grammar:
`VarDecl` $\rightarrow$ `var IDList: typeID`
`IDList` $\rightarrow$ `IDList, ID | ID`
where `IDList` derives a comma-separated list of variable names and `TypeID` derives a valid Pascal type. Write an attribute grammar that assigns the correct data type to each declared variable. Can that scheme operate in a single pass of the syntax tree?

Problem 4 (Cooper and Torczon, Problem 4.3)

Based on the evaluation rules given below, draw an annotated parse tree that shows how the syntax tree for `a - (b + c)` is constructed.

Production	Evaluation Rule
$E_0 \rightarrow E_1 + T$	$\{E_0.nptr \leftarrow mknode(+, E_1.nptr, T.nptr)\}$
$E_0 \rightarrow E_1 - T$	$\{E_0.nptr \leftarrow mknode(-, E_1.nptr, T.nptr)\}$
$E_0 \rightarrow T$	$\{E_0.nptr \leftarrow T.ntpr)\}$
$T \rightarrow (E)$	$\{T.nptr \leftarrow E.ntpr)\}$
$T \rightarrow id$	$\{T.nptr \leftarrow makeleaf(id, id.entry))\}$

Problem 5 (Cooper and Torczon, Problem 4.10)

Object-oriented languages allow operator and function overloading. In these languages, the function name is not always a unique identifier, since it can have multiple related definitions, for example:

```
void show(int attribute);
void show (char * attribute);
void show (float attribute);
```

For lookup purposes, the compiler must construct a distinct identifie for each function. Sometimes, such overloaded functions have different return types as well. How would you create distinct identifiers for such functions?